Syntax and Semantics

Category	Notation	Description/Examples

Primitive Types	`bool`
	`int`
	`float`
	
	`char`
		Takes value `TRUE` or `FALSE`
		Takes positive or negative integer values, including 0
		Takes positive or negative decimal values, of implied infinite precision, including 0.0
		Takes a single character value, including alphanumeric values, non-text characters, or spaces.

Assignment	`=`	Type declaration required for initial assignment, not for later reassignment. `char` types and `String` objects are enclosed by `""` for assignment.
		`int x = 5`
		`x = 6`
		`bool y = TRUE`
		`char initial = "P"`
		`String my_text = "Some text."`
		`int[] my_array = [2, 3, 4]`

Strings	`String`
	
	`String my_pswd = "haggles"`
	`my_pswd = "H@99l35"`
		An ordered `char` sequence. Assigned by surrounding a set of `char` values in double quotes.

Arrays	`[]`
	
	type[] array_name = [row 1 values],
	… [row n values]
	
	array_name[row, column]
		One or two-dimensional, zero-indexed, collections of values of the same type. Two-dimensional access is row-major. One-dimensional arrays or sets have only one row. Access is by `int` index.
		
		`float[] values = [1.1, 2.22, 3.333]`
		
		`String[][] fruits =`
		`[["apple", "banana", "cherry"],`
		` ["avocado", "berry", "cantaloupe"]]`
		
		`my_array[2, 5] = 2.3`

Category	**Notation**	**Description/Examples**

Maps	`{}`
	
	key_type value_type{} name = {key_1: value_1,…key_n, value_n}
	
	name{key} = value
	
	name{key}
		Mutable collections of key-value pairs. Keys must be of a single type. Values must be of a single type. Keys must be unique. Access is by key.
		
		`String String{} book = {"title": "Moby Dick", "author": "Herman Melville", "section": "fiction"}`
		
		Sets value of a key or creates a key with given value
		`book{"year"} = "1851"`
		
		`String writer = book{"author"}`

Mathematics	+ addition
	– subtraction
	* multiplication
	/ division
	^ exponentiation
	% modulo
		Operate on matched type `int` or `float` pairs. Follow standard PEMDAS order of operations.
		`x + b`
		`x - b`
		`x * b`
		`x / b`, `int` pair returns rounded `int` answer
		`x ^ b`
		`x % b`, operates on `int` pairs returns `int` remainder

Relationships	< less than
	> greater than
	<= less than or equal to
	>= greater than or equal to
	
	== equal to
	!= not equal to
		Compare matched matched type `int` or `float` pairs. Expressions evaluate to `bool`.
		`x < b`
		`x > b`
		`x <= b`
		`x >= b`
		Compare type matched pairs of any type. Expressions evaluate to `bool`.
		`x == b`
		`x != b`

Logic	AND all conditions true
	
	OR at least one condition true
	
	NOT condition not true
		Operate on `bool` or expressions that evaluate to `bool`. Parentheses are evaluated first, otherwise operators evaluated left to right
		`(5 < 7) AND (5 == 5) == TRUE`
		`>>> TRUE`
		`(5 > 7) OR (5 == 5)`
		`>>> TRUE`
		`NOT(5 > 7)`
		`>>> TRUE`

Category	**Notation**	**Description/Examples**

Conditionals	if (condition)
	    block
	elseif(condition)
	    block
	else
	    block
		`if (x > 3 AND b < 100)`
		  `String a = "Large enough."`
		`elseif (x >= 100)`
		  `String a = "Too large."`
		`else`
		  `String a = "Not large enough."`

Iterables	while (condition)
	    block
	
	for (iterator; comparison; incrementor)
	    block
		`while (x < 10)`
		  `output(x)`
		  `x = x + 1`
		
		`for (int i = 1; i < 10; i = i + 1)`
		  `output(i ^ 2)`

Functions	function function_name (type arg1,… type argn)
	    block
	
	function_name(value1,…valuen)
		Functions evaluate to the value of the `return` line, if present.
		`function adder(int num_1, int num_2)`
		  `return (num_1 + num_2)`
		
		`z = bin_to_int("10001")`

Comments	/*
	  comment
	*/
		`/*`
		  `some information`
		`*/`

Method/
Attribute
Access
	object.method(args)
	
	object.attribute
		`String my_sentence = "They went away."`
		`output(my_sentence.sub(0,4))`
		`>>> "They "`
		
		`output(my_sentence.len)`
		`>>> 15`

Errors and Null	`ERROR` `NULL`	Invalid or undefined values return or output `ERROR` Searches that produce no result return `NULL`

Classes, Functions, Methods, and Attributes

Category	**Notation**	**Description/Examples**

Built-In Functions	`type(x)`
	`input()`
	`output(y)`
		Returns type of the object or primitive as `String`.
		Accepts user input.
		Outputs to the console. Displayed console output lines begin with `>>>`

`String`	`.len`
	`.sub(int start, int stop)`
	
	`concat(String 1, String 2,…String n)`
		Returns the number of characters as `int`.
		Returns an exclusive `String` from the zero-indexed start to the stop indexes.
		Returns a single `String` produced by combining 2 or more `Strings.`

Arrays	`.size`
	
	`.search(value)`
	
	`.slice(int rowstart, int rowstop, int colstart, int colstop)`
	
	`.insert(index,value)`
	`.append(value)`
	`.del(int index)`
		Returns a two item `int[]` with the number of rows and columns in the array.
		Returns a two item `int[]` with the lowest index of the value . Two-dimensional array search iterates through rows first.
		Returns an array of the same type within the inclusive given index bounds. One-dimensional arrays can implicitly use `rowstart = rowstop = 0`
		
		For one-dimensional arrays:
		Inserts a new value at the given preexisting index.
		Adds a new value to the end of an array.
		Deletes the value at the given index. The indexes of later values decrease by one.

Maps	`.size`
	`.search(value)`
	`.remove(key)`
		Returns the number of key values pairs as an `int`
		Returns an array of the key or keys with the given value.
		Removes a given key and its value from the map.